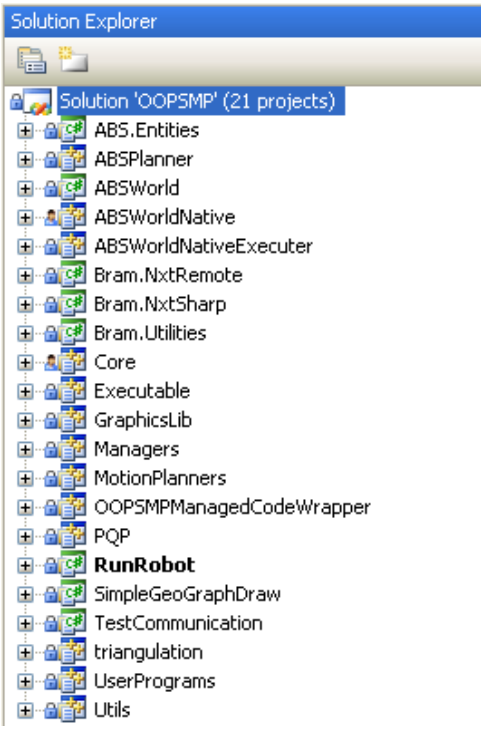
ABS (Avinery Biton Skoran)

Developer Guide



Contents

Projects

Core, Executable, Managers, MotionPlanners, PQP, Triangulation, UserPrograms, Utils:

These are the original OOPSMP library projects. The projects have been removed of their "OOPSMP" prefix, and their files have in some cases been modified, either to be compatible with Visual Studio's build process, to fix bugs, or add small features.

ABSPlanner

This project contains our designed motion planner.
Inside the project are 4 files:

- ABS.cpp/h – The actual motion planner
- ABSParamaterContainer.cpp/h – The motion planner's parameters container

ABS.Entities, ABSWorld, ABSWorldNative, ABSWorldNativeExecuter, GraphicsLib

These projects are a group of projects that handle the debugging user interface.

- ABS.Entities – ".Net" classes that define graphics primitives
- ABSWorld – The user interface main application
- ABSWorldNative – An interface between Native Code (C++) and Managed Code (.Net), that runs the OOPSMP main executer, because OOPSMP's executer is not compiled with ".Net" support
- ABSWorldNativeExecuter – A project in C++ (with ".Net" support), that only serves to work around a disability of Visual Studio – namely, that the debugger cannot debug Native Code (C++) if the executable is in C# (Managed Code).
- GraphicsLib – Serves as a bridge from Native Code to Managed Code for the graphics primitives. The algorithms actually use this library to visualize themselves.

Bram.NxtRemote, Bram.NxtSharp, Bram.Utilities

These projects are a package that enables control over the robot's engines over Bluetooth ™ communication.

RunRobot

This project is responsible for translating the solution path to instructions for the robot.

The project's main code file is: "translateSolutionToGeometricalValues.cs" (contains the entry point)

TestCommunication

This project tests communication over Bluetooth ™ with the robot's control unit.

Main Algorithmic Concept/Flow

Overview (or "General Idea")

- Run several motion planners on the same problem, to get several solution paths
- Create additional solution paths from the original ones, by short-cutting wherever possible
- Create a hybrid solution path from the original paths, using several optimization parameters
- Smooth the hybrid solution
- Translate solution path to robot instructions

Initialization

Just like any other motion planner, with several parameters added.

Solution

- Use RRT, BiTree, PRM, and EST algorithms, some more than once, all with the same local-planner parameters, to solve the given query
- For each solution path produce another one – a shortcut path
- "Fuse" paths by connecting every vertex on all the paths to its K (parameter) unconnected neighbors
- Run a search on the fused-paths graph, using a modified version of Dijkstra's algorithm (one which takes the magnitude of turns into account), with edge-weights which are calculated using the given parameters
    - The given parameters define what is to be optimized, using factors that are used in the sum of the weight:
        - A factor for the edge's length, to optimize **Distance**
        - A factor for the edge's minimal distance from obstacles (multiplied by the length of the edge), to optimize **Clearance**
    - The third optimization – more **efficient turns** – is a weight which is not summed with the other edge weights, but instead is added to the minimal distance to a given vertex, considering the edges that "brought" us to a given vertex
        - When the search algorithm visits a vertex and "relaxes" its neighbors, it "knows" which path from the source got us to the visited vertex, and can calculate the turn from the ingoing edge, to the outgoing edge to the neighbor
        - The turning angle is then multiplied by 10 times the amount of turns raised to the power of 2 [Just to be clear: (10 x TurnsCount)^2], to make sure paths with more turns are treated as much more (more than linearly) difficult than paths with less but sharper turns (this requirement is due to the robot's behavior with turns).
- Produce the hybrid solution path from the search-graph result
- Smooth the solution

Translation to robot commands

- Compute turn angles and length of vertexes from the solution file
- Computes for each turn the distance that the robot "stole" while turning and the distance from the next turning point the robot has to start the turn to complete it in the right spot.
- Establish connection to the robot.
- Sends the commands one by one, while waiting to each command to finish before sending the next one (in normal state- if the drive and turn flag is changed then it sends two commands at a time).

Pre/Post Processing

Due to unexplained issues when solving problems in 3D, and long solving time due to redundant degrees of freedom, we've tried to convert a 3D world, designed for 2D problems, to a 2D world.

The code for that has been implemented in "MotionPlannerProgram" under the "UserPrograms" project.

The solution is then converted back to 3D, in order to be compatible with Google Sketchup's format requirements for the solution file.

The conversion simply ignores any surface not defined on Z=0.

Design Decisions

Avoid "CMake", use Visual Studio's build environment

Since we (at least some of us) have experience with the Visual Studio build environment and compiler, we decided it would be best to use it instead.

The downside of this decision is that we do not guarantee compatibility with the original OOPSMP build process. But we estimate that transforming back to the original build process would not require great effort.

Work inside OOPSMP instead of using it externally as a black box

We've initially contemplated using OOPSMP as a black box and using the different Motion Planners data output for Hybridization and Optimization. We soon realized it would be more beneficial to write our own motion planner so we could gain access to any internal data we might need.

Use Motion Planners solution paths instead of their internal samples

One of our options was to use the internal samples (milestones) and interconnections from different Motion Planners, we then decided that since our goal is optimization, we want to start out with pre-existing solution paths, and use them as a basis for optimization. Naturally, the more solution paths we start with, the more options we have to choose from, and we might get better optimized solutions.

Adding shortcut paths and random paths to the mix

One of the pitfalls of using solution paths as a basis for our algorithm is that motion planners by default use simple path searching algorithms (I.e. DFS), that tend to pass near obstacles and limit our ability to optimize the solution. Bearing in mind that we start out with solutions and generally use parts of them in an optimized way, if we don't have optimized sub-paths to choose from, our final solution would probably not be optimized (though we do through in some tricks of our own).

One way of dealing with this limitation, is by adding new solution paths to the basis of our hybridization algorithm:

- Shortcut paths – paths in which we created shortcuts between vertices on the path, thus supplying the hybridization with straight sub-paths to use

Using Bram project instead of Microsoft Robotics Studio.

When we started looking for a way to control the robot we encountered Bram C# open source project which, after inspection, gives a very easy and convenient interface to the robot (initialize all the ports, communication synchronization and all other aspects of the connection and controlling the robot), While if working with Microsoft Robotics Studio we would have to take care of all those things ourselves.

Important Classes

ABS

This class contains our main algorithm's code. It is defined in the ABS.cpp/h files in "ABSPlanner" projects.

The class contains important methods which we've added:

- "Hybridization" – handles the hybridization process
- "DoShortcuts" – produces shortcut paths from original paths
- "SmoothHybridPath" – Causes "fine" smoothing by limited, non-localized, shortcuts on the final hybrid path, with a limitation on the deviation from the original path
- "localizedShortcuts" – Causes localized shortcuts on the final hybrid path, does not limit deviation from the original path

Most other methods are overloaded from the parent class "MotionPlanner".

ABSParamaterContainer

The class contains the parameters for the ABS class.

SearchGraphEdgeWeightAsCost

Each edge has an attached data. A search algorithm uses a generic cost calculating object. This class returns the edge's weight, which is encapsulated in the edge's attached data, to the search algorithm as its (the edge's) cost.

Rand

This is a utility class that provides us a simple pseudo-random algorithm, used to calculated random weights based on vertex id (in order to provide consistent "random" weights).

Remarks

Exporting DLL functions/classes

In transforming parts of the code to comply with Visual Studio's build process; many classes had to be prefixed with a "DLL Export" instruction.

Graphic Motion Planners dropped

Since we only adapted to Visual Studio's build-process the code we needed, we did not adapt the Graphic Motion Planners as well. Some small work is needed to adapt that code, but we did not find a necessity for that.

Memory Leaks

We know our code "leaks" memory. Since the memory consumption involved is not an issue for modern computers, we've decided not to address the issue in the time constraints we had, because when we sometimes cleaned-up our memory ("delete") we caused corruption of data we were still using.

Linking Google Sketchup with our executable

In order to call our executable instead of OOPSMP's, and to gain more freedom to modify parameter etc., we got OOPSMP's plugin to Sketchup to run a batch file which we can modify more easily than the plugin's code. The batch file is located in Sketchup's folder and is names "run_oopsmp.bat".

External Code Packages

OOPSMP

An Object-Oriented Programming System for Motion Planning

http://www.kavrakilab.org/OOPSMP/index.html

Bram nxt

C# code that communicates with the Lego NXT computer and exposes methods to control the robot.

http://nxtsharp.fokke.net/